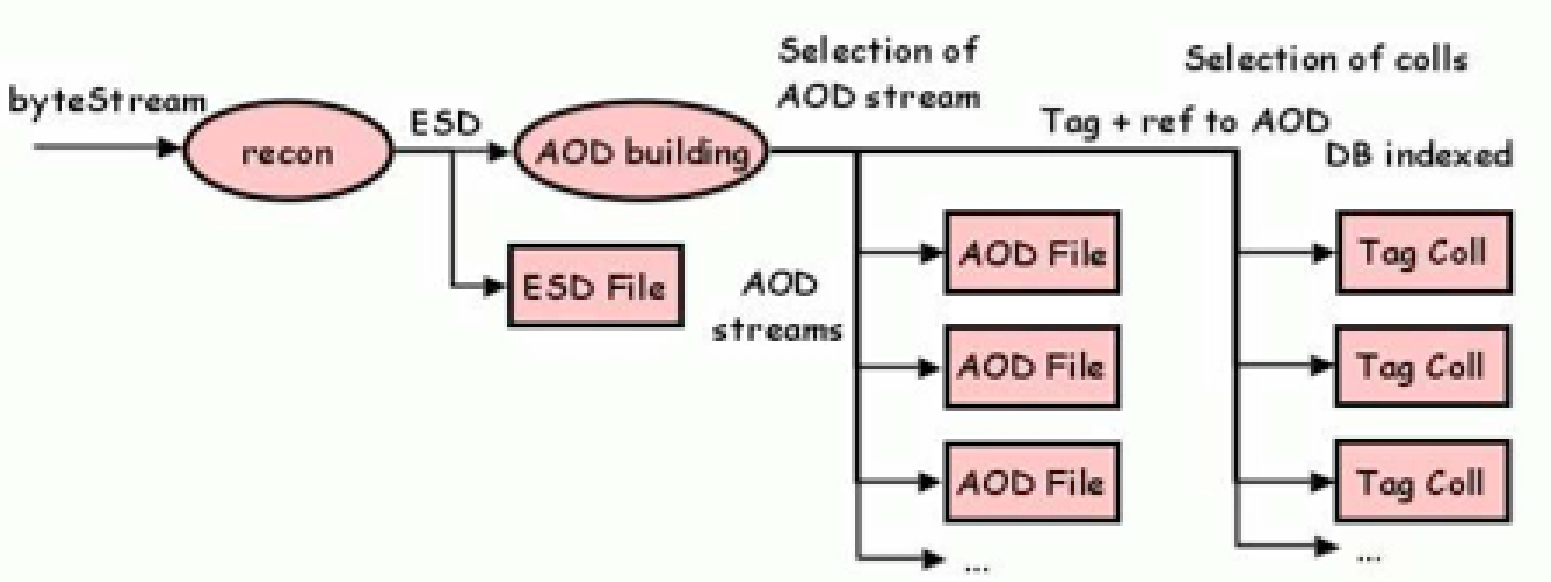
AOD's Tutorial

Simona Rolli

Outlook

- ATLAS even data model
  - ESD/AOD contents
- Athena framework
- How to access AOD
  - Example code
- Highlights from performance groups

ATLAS even data model



Three major components in addition to RAW data:
ESD - detailed reconstruction output (500kB)
AOD - compressed form meant to be used by most analyses (100kB)
TAG- relational db entries for quick event preselection (1 kB)

ESD/AOD architecture

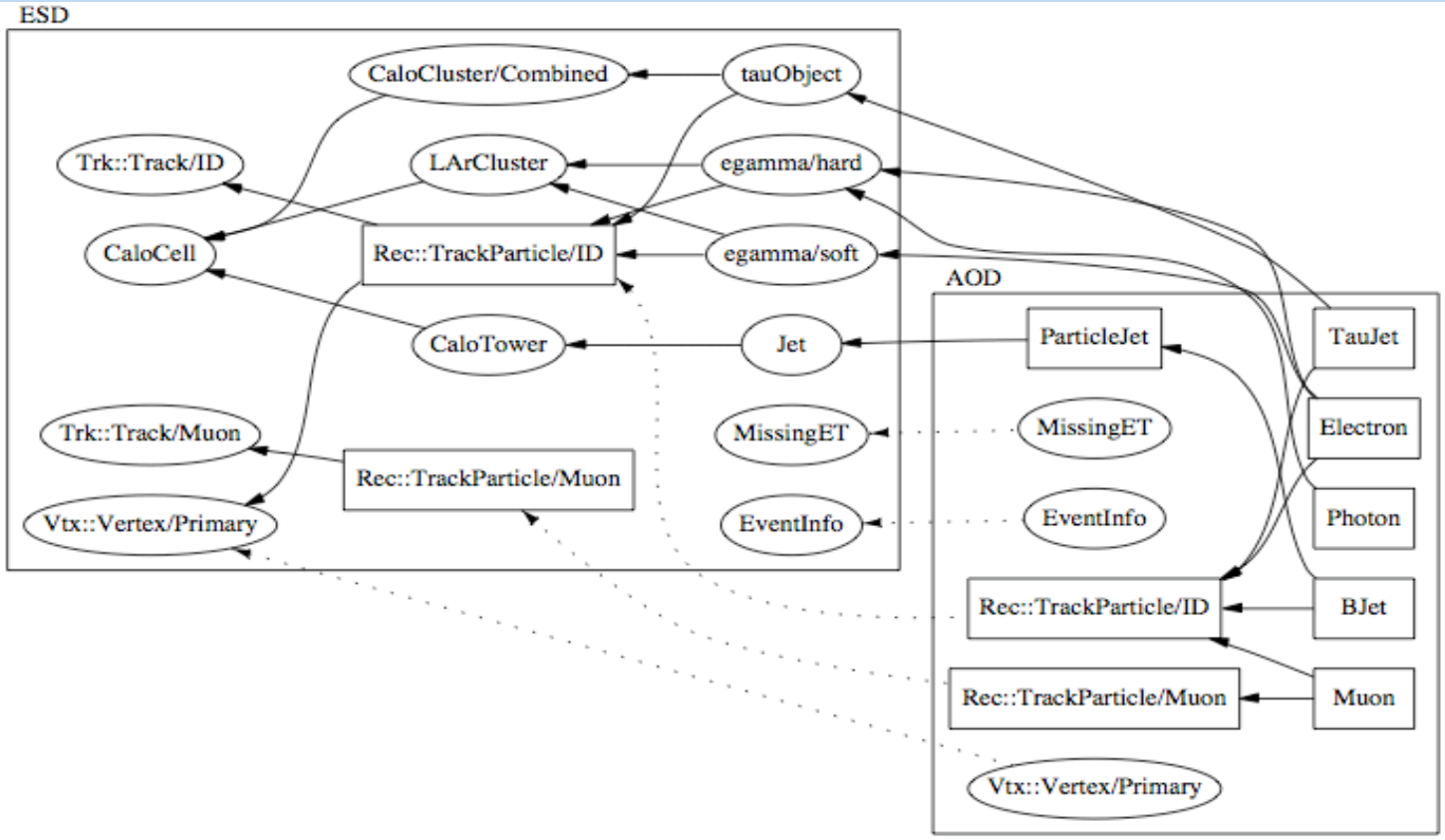


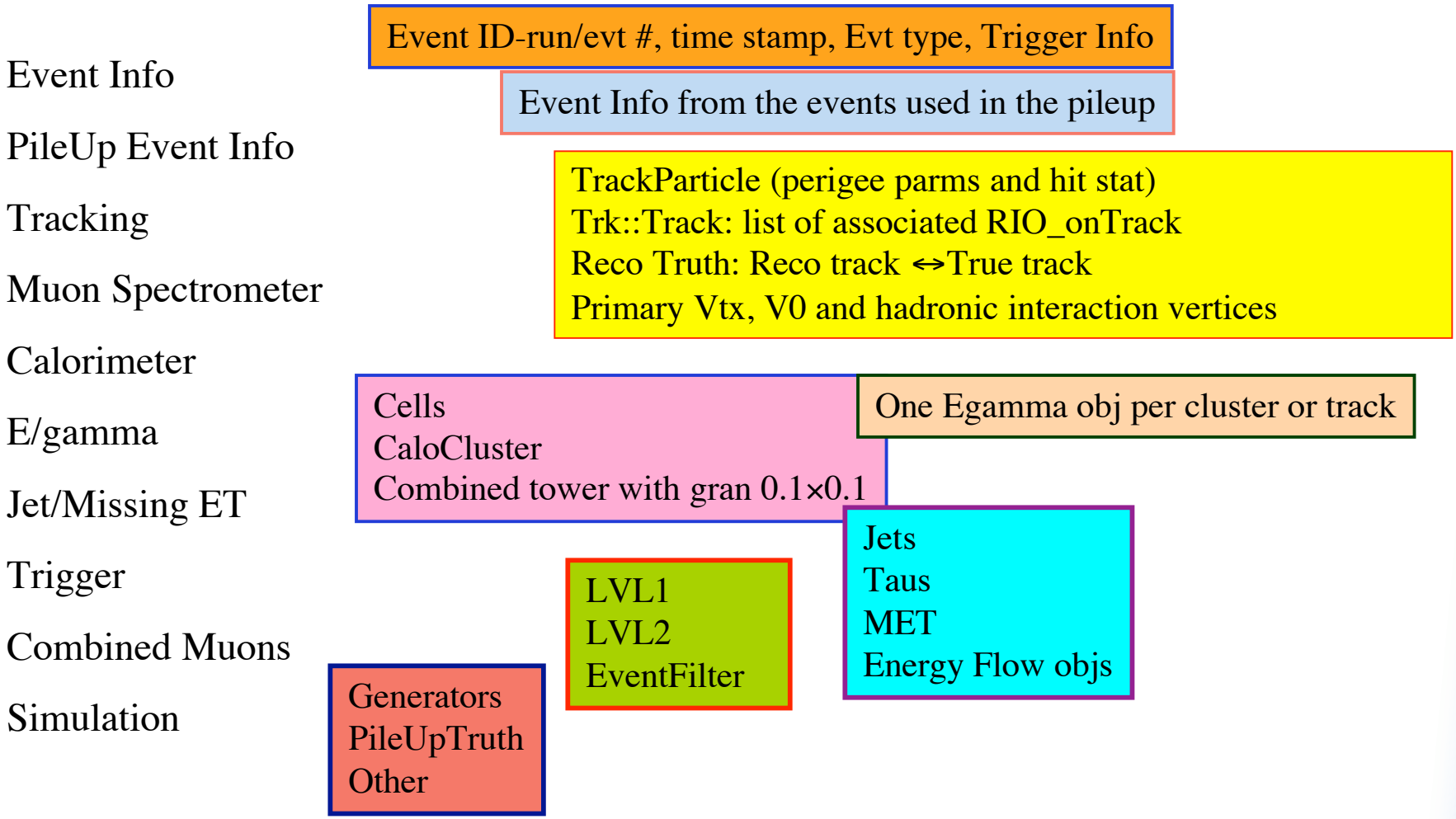
Figure 2: Main objects in ESD and AOD. Only objects collection of which are available in StoreGate are indicated. Some auxilliary objects have been omitted. Multiple instances of same objects are often omitted for clarity. Solid arrows indicate "pointed to" relationship. Dotted arrows indicate "copied from" relation ship. Object implementing the IParticle interface are indicated with square boxes. All such objects in AOD points to the primary vertex in AOD, the corresponding arrows have been omitted for clarity.

ESD/AOD Architecture

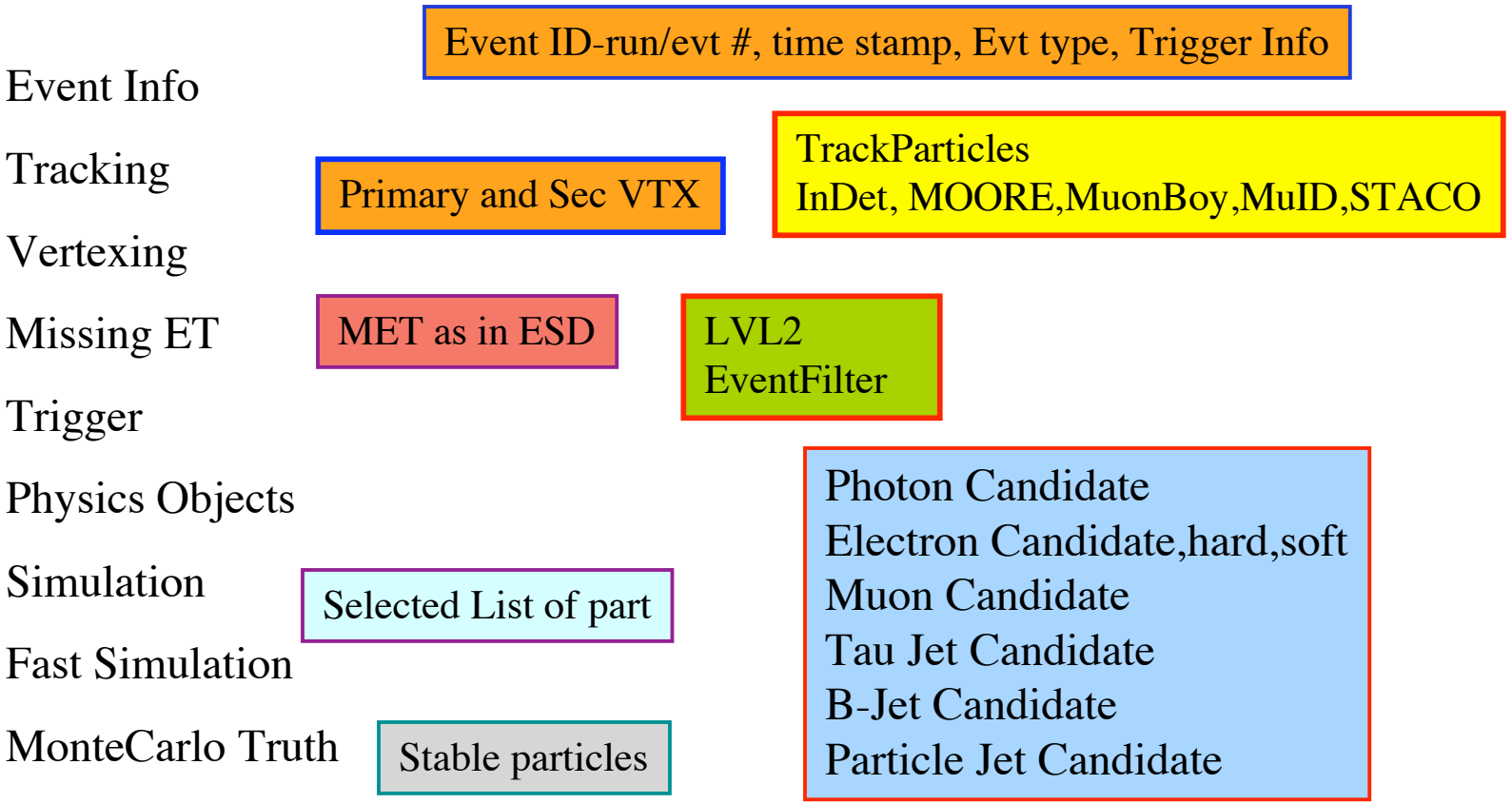
3 cases for reconstruction:

1) Algs which have output too large to fir AOD or can not be run from ESD → ESD dedicated object + AOD reduced object.
   - example: egamma(ESD) and Electron and Photon (AOD)
2) Reconstruction algs can be run on ESD→ the algs are run as part of the AOD making process
   - example: btagging, Staco
3) Reconstruction algs cannot be run on ESD, output doe fit AOD size → run in the ESD making step. *Transitory*

ESD Content


Event Info
PileUp Event Info
Tracking
Muon Spectrometer
Calorimeter
E/gamma
Jet/Missing ET
Trigger
Combined Muons
Simulation
Event ID-run/evt #, time stamp, Evt type, Trigger Info
Event Info from the events used in the pileup
TrackParticle (perigee parms and hit stat)
Trk::Track: list of associated RIO_onTrack
Reco Truth: Reco track ↔True track
Primary Vtx, V0 and hadronic interaction vertices
Cells
CaloCluster
Combined tower with gran 0.1×0.1
One Egamma obj per cluster or track
Jets
Taus
MET
Energy Flow objs
LVL1
LVL2
EventFilter
Generators
PileUpTruth
Other

AOD Content


Event Info
Tracking
Vertexing
Missing ET
Trigger
Physics Objects
Simulation
Fast Simulation
MonteCarlo Truth
Event ID-run/evt #, time stamp, Evt type, Trigger Info
Primary and Sec VTX
TrackParticles
InDet, MOORE,MuonBoy,MuID,STACO
MET as in ESD
LVL2
EventFilter
Photon Candidate
Electron Candidate,hard,soft
Muon Candidate
Tau Jet Candidate
B-Jet Candidate
Particle Jet Candidate
Selected List of part
Stable particles

AOD Keys

```
Event Info                          = "McEventInfo"
ElectronContainer Name              = "ElectronCollection"
PhotonContainer Name                = "PhotonCollection"
MuonContainer Name                  = "MuonCollection"
TauJetContainer Name                = "TauJetCollection"
JetTagContainer Name                = "BJetCollection" (since 10.0.0)
ParticleJetContainer Name           = "ParticleJetContainer"
(up to 9.4.0/9.0.4)
ParticleJetContainers (since 10.0.0) = "KtTowerParticleJets",
                "Cone4TowerParticleJets", "ConeTowerParticleJets"


Missing Et objet Name               = "MET_Base"
Missing Et calibrated objet Name    = "MET_Calib"
Missing Et Truth objet Name         = "MET_Truth"
Missing Et Muon  objet Name         = "MET_Muon"
Missing Et Final objet Name         = "MET_Final"
Missing Et Cryostat correction      = "MET_Cryo"   (since 10.0.0)
Missing Et Topological Clusters     = "MET_Topo"   (since 10.0.1)
TruthParticleContainer Name         = "SpclMC"     (filled on the fly)
```

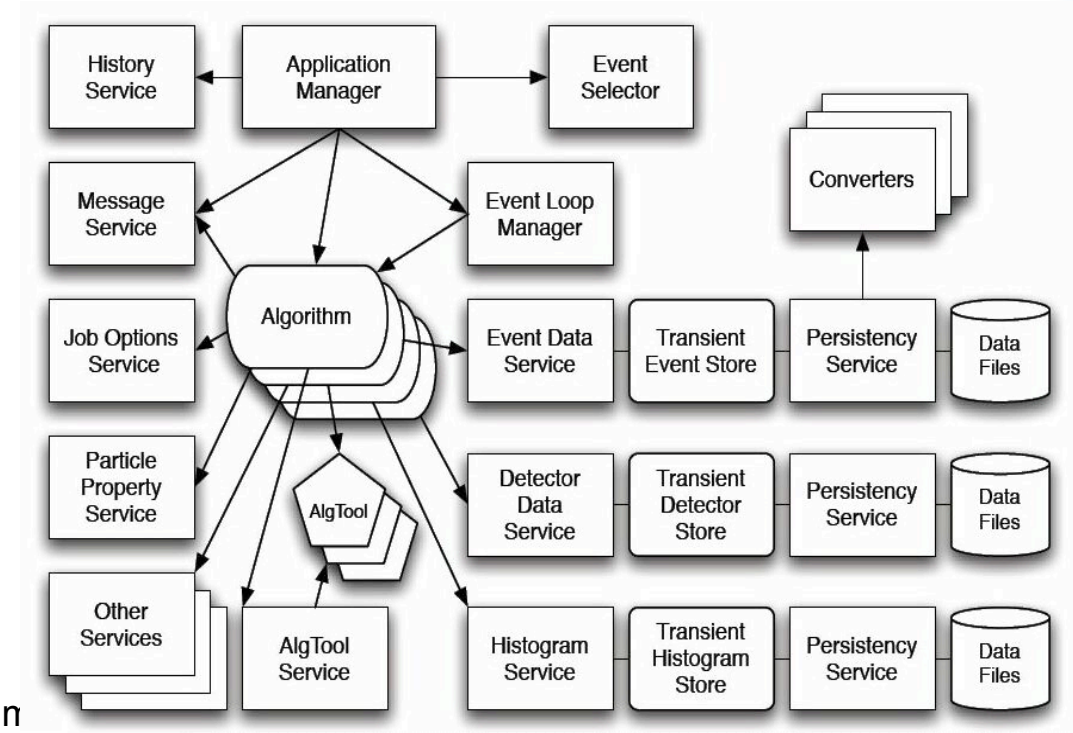
AOD Keys (II)

```
Inner Detector TrackParticles        ="TrackParticleCandidate"
Inner Detector TrackParticles        = "TrackParticleCandidateXK"
Muonboy TrackParticles               = "MuonboyTrackParticles"
STACO TrackParticles                 = "StacoTrackParticles"
MOORE TrackParticles                 = "MooreTrackParticles"
MuID StandAlone (SA) TrackParticles = "MuidStandAloneTrackParticles"
MuID Combined TrackParticles         = "MuidCombnoSeedTrackParticles"
Muid SA TrackParticles low Pt        = "MuidStandAloneTrackParticlesLowPt"
Muid MOORE TrackParticles low Pt     = "MuidMooreTrackParticlesLowPt"
Muid Combined TrackParticles low Pt = "MuidMooreTrackParticlesLowPt"
Muid iPatTrackParticles              = "MuidiPatTrackParticles"
Muid iPatTrackParticles low Pt       = "MuidiPatTrackParticlesLowPt"
Slimmed McEventCollection            = "GEN_AOD"
TrackParticleTruthCollection         = "TrackParticleTruthCollection"
CTP Decision                         = "CTP_Decision"
LVL1 RoI                             = "LVL1_ROI"
Vertex Container                     = "VxPrimaryCandidate"
Track Record Collection              = "MuonEntryRecordFilter"
Truth ParticleJetContainers (since 10.0.0)   = "KtTruthParticleJets",
"Cone4TruthParticleJets", "ConeTruthJets”
```

A reminder about ATHENA

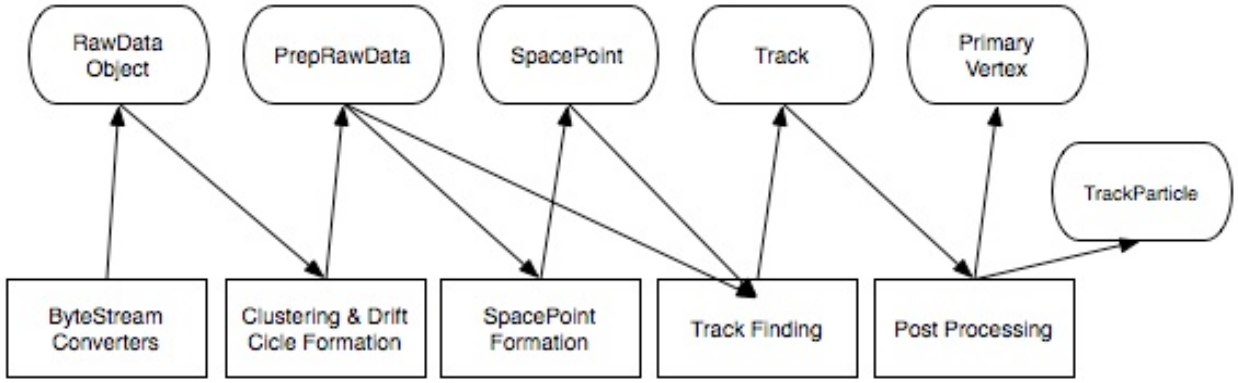
- The architecture of the Athena framework allows for:
  - Separation of data from algorithms
  - Separation of transient (in-memory) from persistent (in-file) data
  - Extensive use of abstract interfaces to decouple the various components

- Backbone of the ATLAS computing system
- Quite extensively used
- Field tested (CTB)
- It scales, it works…

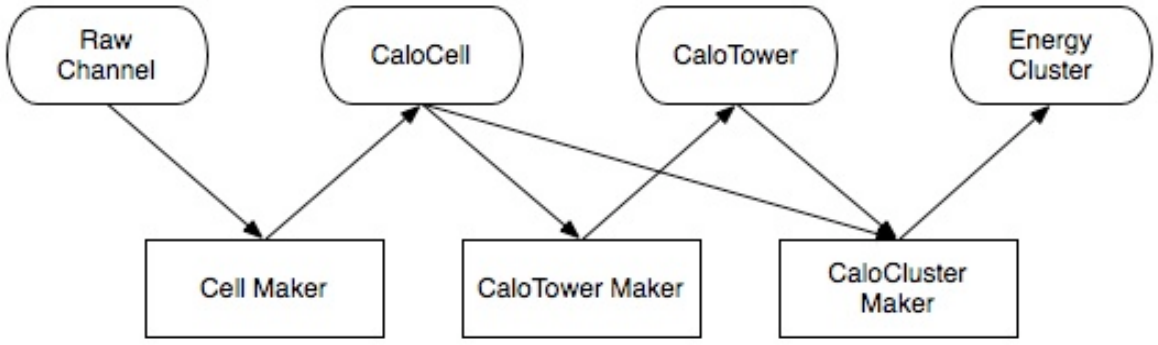


Separation of data and algs

- Tracking code:



- Calorimetry code:



Example Code

There is an excellent tutorial on how to setup an analysis code to run on AOD/ESD by Ketevi Assamagan:
http://www.usatlas.bnl.gov/PAT/analysis_on_aod.html#Ana_AOD

I have followed his example in setting up the user code you might have found in the twiki page, to read AOD/ESD and produce a simple ntuple.

What follows is taken from Ketevi's tutorial, almost verbatim

Code Example: UserAnalysis

The CVS Package UserAnalysis

This is the proposed package where the user will develop his/her analysis code, UserAnalysis.
After setting up CMT, check this package out of the CVS repository into your working directory:

cmt co PhysicsAnalysis/AnalysisCommon/UserAnalysis
or
cmt co -r UserAnalysis-ab-cd-xy PhysicsAnalysis/AnalysisCommon/UserAnalysis
where ab-cd-xy is a specific package tag number such as 00-01-09.

This package provides a default requirements file to build the user analysis code and an analysis skeleton algorithm class called AnalysisSkeleton.
All the user has to do is to rename this class to his/her liking and start developing the analysis code.

Code Example: compiling, linking and running

You can compile and link the analysis skeleton algorithm and run it:
To do this, as follows:
build the AnalysisSkeleton algorithm with the following commands from the "cmt" directory of UserAnalysis

```
cmt config
source setup.(c )sh
cmt broadcast gmake
```

go to the "run" directory of UserAnalysis and do this

```
get_files AnalysisSkeleton_jobOptions.py

athena.py AnalysisSkeleton_jobOptions.py
```

Although the above command is correct, it will most likely not run because the input AOD file and the corresponding PoolFileCatalog.xml are not defined.

Code Example: input file

One would then need to do the following:

edit this job options for the AOD input file name and location

Make sure PoolFileCatalog.xml exists and contains the AOD input file,

Then
athena.py AnalysisSkeleton_jobOptions.py

This should run the AnalysisSkeleton algorithm producing some ROOT histograms in a file named: AnalysisSkeleton.root

Code Example: our example

On the twiki page you'll find the instructions to run AnalysisSkeleton renamed DoubleChargedHiggsOnAOD
The name is from the code Kamal Benslama gave me and I have been to lazy to rename it!

So, what does the code do? What information is accessed from AOD (ESD) ?

Back on AOD objects

You need to know the "name" (or the StoreGate key) of the AOD container that you want to access in your analysis code.
What we mean by name here is the std::string code that was used to create the AOD container. You need to know that so you can ask for that container if you need it.

```
std::string m_electronContainerName = "ElectronCollection";
...  const ElectronContainer* elecTES;
sc=m_storeGate->retrieve( elecTES, m_electronContainerName );
if( sc.isFailure()  ||  !elecTES ) {
mLog << MSG::WARNING
<< "No AOD electron container found in TDS"
<< endreq;      return StatusCode::SUCCESS;  }
else{
mLog << MSG::DEBUG << "ElectronContainer successfully retrieved"
<< endreq; }
```

In the above snippet of code, the data member m_electronContainerName defines the name of the ElectronContainer that you want to retrieve. If you look in the constructor of AnalysisSkeleton.cxx, you will see that m_electronContainerName is initialized to "ElectronCollection".
That is the name used when the AOD electron container was produced

AOD interfaces

Note that from the link to the container class, you can acccess the class of the contained object.
For example, from the link to ElectronContainer.h,
you can access Electron.h which is the electron AOD class.

The common implementation for some of the AOD classes is provided in the ParticleBase.h class (charge, Id, PartType).
The navigation features are implemented in Navigable.h (technical)
and the four momentum interface is given by 4Mometum (package with several classes to handle different requirements)

Common way to access information!

The common tools, to handle AOD objects in your analysis, are built on the ParticleBase.h and the ParticleBaseContainer.h interface.
(http://www.usatlas.bnl.gov/PAT/analysis_on_aod.html#Tools_AOD)

Accessing objects inside a container

Loop over the objects in the Container: after you successfully retrieve the container and have a pointer to it in the transient data store (TDS),
you can get the iterators over the container since the container is a DataVector.
Look at the example in the execute() method of AnalysisSkeleton.cxx:

```
...  /// iterators over the container
ElectronContainer::const_iterator elecItr  = elecTES->begin();
ElectronContainer::const_iterator elecItrE = elecTES->end();
    for (; elecItr != elecItrE; ++elecItr) {
       if( (*elecItr)->hasTrack() &&
           (*elecItr)->pt()> m_etElecCut ) {
              m_h_elecpt->fill( (*elecItr)->pt(), 1.);
              m_h_eleceta->fill( (*elecItr)->eta(), 1.);
...
```

In the above snippet of code, (*elecItr) is a pointer to an Electron.h within the container ElectronContainer.h

Access to kinematics

Some of the AOD objects are 4Momentum objects,
meaning that they should be able to answer all your questions about their kinematics.
For example, to ask the Electron object for its transverse momentum and
pseudo rapidity:

```
for (; elecItr != elecItrE; ++elecItr) {
if( (*elecItr)->hasTrack() && (*elecItr)->pt()> m_etElecCut )
{         double electronPt  =  (*elecItr)->pt();
          double electronEta = (*elecItr)->eta();
...
```

Kinematics information

The complete list is the following:

```
double px()
double py()
double pz()
double m()
double p()
double eta()
double phi()
double e()
double et()
double pt()
double iPt()
double cosTh()
double sinTh()
double cotTh()
HepLorentzVector hlv()
```

Performances groups

In ATLAS there are 4 groups working in definition and testing the Performances of the physics objects:

1. e/gamma
2. Jet/Tau/MET
3. b-tagging
4. Muons

Here we report on recent highlights from last week overview in Paris

e/gamma reconstruction

The reconstruction sequence for electrons and photons:

- Calibration of Electronics and Alignment
- Clustering either Topological or Sliding Window
  SW used in the following
- Corrections at the cluster level:
  - position corrections
  - correction of local response variations
  - corrections for losses in upstream (Inner detector) material and longitudinal leakage
- Matching with Tracks
- Identification
- 2nd stage reco:
  - Refinement of corrections depending on the particle type (e/γ) planned
  - Bremfit/Gaussian Sum Filter planned
- uniformity 0.7% with a local uniformity in $\eta X \phi$=0.2x0.4 better than 0.5%
- inter-calibrate region with Zee
- AOD: electrons are merged from 2 algorithms, photons are track-less (as orthogonal as currently possible)

e/gamma: ID cuts

There are 3 types of quality cuts you can perform on the electron candidates:

1. Cuts based on the `isEM` flag
2. Cuts based on likelihood
3. Cuts based on NeuralNet output

1. The `isEM` flag uses both calorimeter and tracking information in addition to TRT information. The flag is a bit field which marks whether the candidate passed or not some safety checks. The bit field marks the following checks:

```
Cluster based egamma
ClusterEtaRange          =  0,
ClusterHadronicLeakage   =  1,
ClusterMiddleSampling    =  2,
ClusterFirstSampling     =  3,
Track based egamma
TrackEtaRange            =  8,
TrackHitsA0              =  9,
TrackMatchAndEoP         = 10,
TrackTRT                 = 11
```

In 9.0.4 there is a problem with TRT simulation so one has to mask TRT bit to recover the lost efficiency.
To get the flag in your AOD analysis you should use:
`(*elec)->isEM()`
To mask the TRT bits you should use:
`(*elec)->isEM()&0x7FF==0`
If you use `isEM` then you will select electrons with an overall efficiency of about 80% in the barrel but much lower in the crack and endcap.

e/gamma: likelihood

The likelihood ratio is constructed using the following variables:
energy in different calorimeter samplings, shower shapes in both eta and phi and E/P ration. No TRT information is used here.
You need to access two variables called `emweight` and `pionweight` then you can construct the likelihood ratio, defined by:
`emweight/(emweight+pionweight).`

In AOD, you use the following code:

```
ElecEMWeight =
elec*->parameter(ElectronParameters::emWeight);
ElecPiWeight =
elec*->parameter(ElectronParameters::pionWeight);
```

Then form the variable:

```
X = ElecEMWeight/(ElecEMWeight+ElecPiWeight);
```

Requiring X > 0.6 will give you more than 90% efficiency for electrons.

e/gamma: neural net

The NeuralNet variable uses as inputs the same variables used for likelihood.

To use it in AOD you should proceed as follow:

```
ElecepiNN = elec*->parameter(ElectronParameters::epiNN);
```

Requiring ElecepiNN > 0.6 will give you about 90% eff for electrons.

However, you should be aware that the NN was trained in full eta range while the likelihood was computed in 3 bins in eta: barrel, crack and endcap.
So I would suggest to use likelihood for now.To require an isolated electron, you have to cut on the energy deposited in the cone around the electron cluster. ATLFAST for example requires Et<10 GeV in a cone of dR=0.2.
You can simulate the ATLFAST cut by requiring `etcone20<10.*GeV`

Jets

The jet reconstruction algorithms implemented in ATHENA are: seeded and seed-less cone and kT algorithms.

These algorithms can act on calorimeter towers ($\Delta\varphi x\Delta\eta$ regions), MC particles or TopoClusters (energy blobs).

Jet reconstruction SW allows to easily change from one input to another using exacly the same reconstruction algorithm.

Mostly seeded cone algorithm – fast, easy to understand however it does not satsify theory requirements (infrared and collinear safety).

Systematic work to understand detailed efficiencies, jet shapes and peculiarities of various algorithms is starting now.

ETmiss reconstruction and calibration

MET requires complete reconstruction and calibration of the event.

MET = MET_Calib + MET_Muon + MET_ Cryo

All Calorimeter cells $|\eta|<5$

calibrated with H1

$|Ecell| > 2\sigma$

OR

All Calorimeter cells belonging to topoclusters 4-2-0

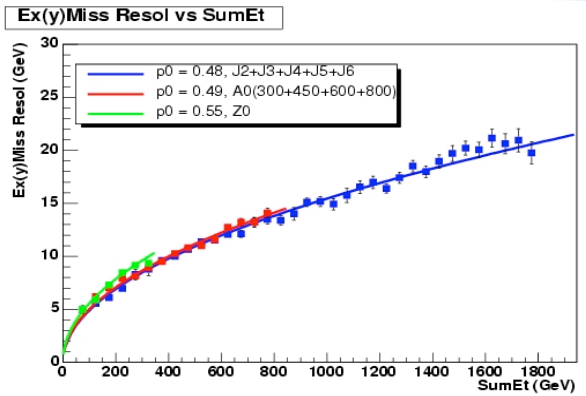
calibrated with H1

All MOORE (to avoid energy double counting) muons

$|\eta| < 2.5$

Estimated energy loss in cryostat between LAr and Tile

Best variable: MET_Final
or a vector sum of
(MET_Topo+MET_Cryo+MET_Muon)



A closer look at ETmiss: tails

- In SUSY working group, started to look at ETmiss tails .

QCD events with a large reconstructed ETmiss are a very dangerous background for SUSY.

- In QCD with 560<pT<1120 (J6) found few hundred events with very large MET found two causes for these … maybe three:

1. There are cells in LArg strips with very large negative energy giving a large NEGATIVE (unphysical!) SumEt. This is a known problem in some Rome events (only in some samples and <3%) in the simulation/digitisation phase. Still to be completely understood.
2. Events with very large pt fake muons (up to % in very high pT jet samples) . Problem is being fixed trying to use better muon reconstruction. Match of muon reconstructed in the spectrometer alone to the ID muon with cleaning cuts.
3. Also few LArg cells with very large positive energy: problem under investigation

4. Deep investigation is continuing….

Muons

Muon identification basics:

I Muon system as tracker (available)

- standalone Muon system
  Moore and Muonboy
- combination of muon and ID track (calorimeter)
  MuId and STACO

II Muon system as tagger (under development)

- starting point is ID track
- muon identification by extrapolation and matching to muon segments
  - Low pt algorithm: hits
  - MuTag: segments

Need less info: not a Track, just a segment
Less sensitive to alignement

Muon access

The muons have highPt and lowPt algorithms.
The overlap is removed, but you may want to only use the highPt ones.

The `chi2()` method is always 0 in 10.0.1, so you will have to access the CombinedMuon through something like

```
const Rec::TrackParticle* cbndMuon =
part->get_CombinedMuonTrackParticle();
 if( cbndMuon ) {
double chi2 = cbndMuon->fitQuality()->chiSquared();
  int ndof =  cbndMuon->fitQuality()->numberDoF();
 if( ndof > 0 ) chi2 = chi2/ndof;
 return chi2; }
```

b-Tagging

- Historical » taggers:
  - **IP2D**: transverse impact parameter
  - **IP3D**: 2D+longitudinal
  - **SV1, SV2**: inclusive secondary vertex **SV1+IP3D** (called SV1 in CBNT)
- New taggers:
  - Lifetime2D: transverse impact parameter
  - **IhSig**: secondary vertex + impact parameter (2D&3D)
- Tagging weight:
  - IP2D: based on impact parameter significances $S=d_0/\sigma(d_0)$
  - Track weight: likelihood ratio $w_t=P_b(S)/P_u(S)$
  - Jet weight: $W_j= \Sigma \ln w_t^i$
- Generalization of the weight for other taggers, can be combined by summing them up.

b-Tagging

- Weights accessed from AOD:
  - M_bjetwSV1[j] = (*newBJets)[j]->weightForTag("SV1");
  - m_bjetwIP2D[j] = (*newBJets)[j]->weightForTag("IP2D");
  - m_bjetwIP3D[j] = (*newBJets)[j]->weightForTag("IP3D");
  - M_LHSig[j] = (*newBJets)[j]->Lhsig();

✓ All taggers are kept for performance studies and cross-checks
⇒ low performance taggers (Lifetime2D/IP2D) are usually rather robust
(easier to understand and commission)
⇒ high performance ones (SV1/SV2) will require more time to control
⇒ taggers identical wrt discriminating variables
(Lifetime2D ~ IP2D, Lifetime3D ~ IP3D)
are kept for cross-checks and do differ in some point
(refined track selection in IPxD,
one 2D vs one 1D pdf for IP3D vs Lifetime3D, …)

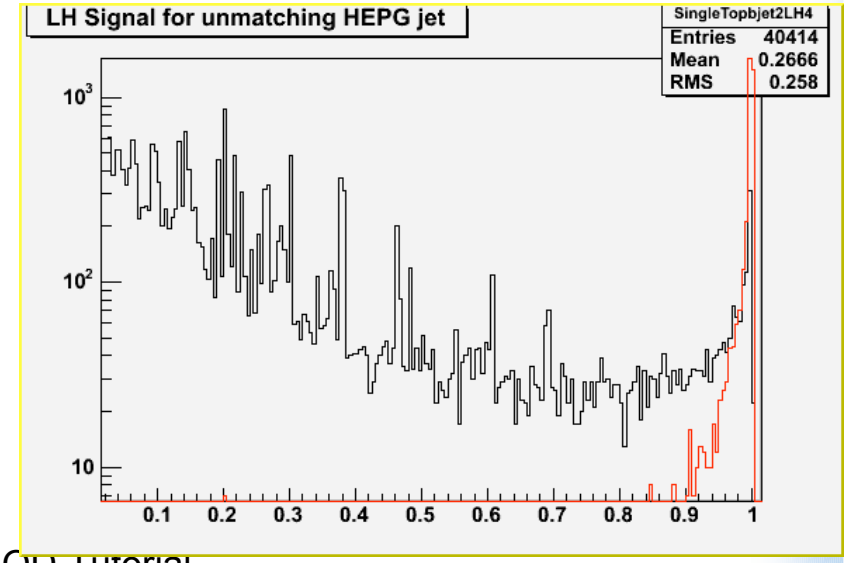
b-Tagging

For physics analysis a combination is given:

- "1st stream" taggers : (*JetTag)→weight()
  corresponding to SV1+IP3D
  The most powerful tagger

- "2nd stream" taggers : (*JetTag)→weightForTag("lhSig")
  corresponding to Lifetime1D+Lifetime2D(+SecVtxBU)

LHSig distribution:
IP2D > 3.0 (red)
IP2D < 1.0 (black)



Conclusions

Analysis Objects Data are quite well defined to start using them

There are several very well documented pages on how to access them and produce ntuple:

- Ketevi's tutorial
- NikHEF page

Following the examples should not be too difficult

Don't be afraid to ask